We claim:

1. A computerized method for creating an instrumented executable file, the method comprising:

redirecting an original function in an executable file to a user-supplied function; and

retaining access information of the original function.

2. The computerized method for creating an instrumented executable file as in claim 1, wherein the user-supplied function is modified to invoke the original function using the retained access information of the original function.

3. The computerized method for creating an instrumented executable file as in claim 1, wherein the user-supplied function is in a dynamic linked library.

4. The computerized method for creating an instrumented executable file as in claim 1, wherein the user-supplied function is not exported during compilation.

5. The computerized method for creating an instrumented executable file as in claim 1, wherein the original function and the user-supplied function have identical prototypes.

6. The computerized method for creating an instrumented executable file as in claim 1, where the user-supplied function is stored in a module that is separate from the executable file.

7. The computerized method for creating an instrumented executable file as in claim 1, wherein the redirecting further comprises modifying the executable file.

8. The computerized method for creating an instrumented executable file as in claim 7, wherein modifying the executable file further comprises determining whether the original function implements the thiscall calling convention, and when the determination is positive, adding instructions to the executable file to perform:

pushing the register that holds the 'this' pointer onto the stack from the invoked original function site when the determining indicates that the function implements a thiscall calling convention; and

swapping the return value of the invoking original function on the stack and the register that holds the 'this' pointer value on the stack when the determining indicates that the function implements a thiscall calling convention.

9. The computerized method for creating an instrumented executable file as in claim 7, wherein modifying the executable file further comprises enabling the user-supplied function to invoke the function in the executable file.

10. The computerized method for creating an instrumented executable file as in claim 9, wherein enabling the user-supplied function to invoke the original function in the executable file further comprises:

adding a jump in the user-supplied function to a function that retrieves the address of the original function; and

adding a jump in the user-supplied-function that invokes the original function using the address of the original function.

11. The computerized method for creating an instrumented executable file as in claim 1, further comprising enabling the user-supplied function to alter behavior.

12. The computerized method for creating an instrumented executable file as in claim 11, wherein enabling the user-supplied function to alter behavior is performed in response to data.

13. The computerized method for creating an instrumented executable file as in claim 12, wherein the data is retrieved from an initialization file.

14. The computerized method for creating an instrumented executable file as in claim 1, wherein the retaining further comprises:

saving the address of an original function in a threaded local storage variable; and

creating an entry in a function lookup table associating the address of the original function with the name of the original function, wherein the function lookup table is in the instrumented executable file.

15. A computerized method for executing an instrumented executable file, the instrumented executable file having an original function redirected to a user-supplied function, and the user-supplied function having a jump to the original function, the method comprising:

saving the address of the original function in a threaded local storage variable; and

invoking the user-supplied function.

16. The computerized method for executing an instrumented executable file as in claim 15, further comprising creating a master lookup table at initialization wherein the master lookup table associates the base address of the instrumented executable file to the address of a function lookup table in the instrumented executable file.

17. The computerized method for executing an instrumented executable file as in claim 15:

wherein original function is in a dynamic link library; and

wherein the saving and the invoking is performed by a stub function of the original function, the stub function being located in the instrumented executable file.

18. The computerized method for executing an instrumented executable file as in claim 15:

wherein original function is embedded in the instrumented executable file; and

wherein the saving and the invoking is performed by the original function.

19. The computerized method for executing an instrumented executable file as in claim 15, further comprising invoking the original function from within the user-supplied function using the threaded local storage variable.

20. The computerized method for executing an instrumented executable file as in claim

19, wherein invoking the original function further comprises:

pushing the register that holds the 'this' pointer onto the stack from the invoked original function site when the determining indicates that the function implements a thiscall calling convention; and

swapping the return value of the invoking original function on the stack and the register that holds the 'this' pointer value on the stack when the determining indicates that the function implements a thiscall calling convention.

21. A computerized method for instrumenting an imported function in an executable file for testing by callers of the imported function, the method comprising:

adding a wrapper of the imported function to an import data block;

adding a stub function for the imported function wherein the stub function comprises and instruction that saves the original function address to a threaded local storage variable and an instruction that causes a jump to the user-supplied function; and

adding an entry in a function lookup table of the original imported function.

22. The computerized method for instrumenting an imported function in an executable file as in claim 22, the method further comprising:

determining if the prototype of the imported function is correctly specified; and

indicating an error when the determining indicates an incorrectly specified prototype of the imported function.

23. A computerized method for instrumenting an embedded function in an executable file for testing by callers of the embedded function, the method comprising:

redirecting an embedded function to the wrapper; and

adding an entry in a function lookup table of the address of the embedded function.

24. A computerized method for instrumenting an embedded function in an executable file as in claim 23, wherein the redirecting is accomplished by an instruction that causes a jump to the user-supplied function.

25. A computerized method for instrumenting an embedded function in an executable file as in claim 23, the method further comprising:

determining whether the prototype of the embedded function is correctly specified; and

indicating an error when the determining whether the prototype of the embedded function is correctly specified indicates an incorrectly specified prototype of the embedded function.

26. A computerized method for instrumenting an embedded function in an executable file as in claim 23, wherein the function lookup table is in the executable file.

27. A computerized method for publishing a function, the method comprising adding an entry describing the function in a function lookup table in a machine-readable executable file.

28. A computerized system comprising:

means for redirecting an original function in an executable file to a user-supplied function; and

means for retaining access information of the original function.

29. A computerized system comprising:

an executable file having a call to an original function, the original function having an identity comprising a name and a parameter prototype; and

means for a user-supplied function that includes a call to the original function to retrieve stored access information of the original function.

30. A computerized system comprising:

an executable file having a jump to an original function, the original function having an identity comprising a name and a parameter prototype;

a first software component having a user-supplied function that includes a jump to the original function; and

a second software component for:

receiving the identity of the original function;

receiving the identity of the user-supplied function;

instrumenting the function in the executable file using the identity of the user-supplied function; and

storing the original function address in the executable file in association with the name of the original instrumented function.

31. A computerized system comprising:

a first module of machine-readable code comprising:

a first instrumented function call to a first replacement function; and

a first data structure associating the identity of the first instrumented function with the location of the first instrumented function; and

a second module comprising the first replacement function, operatively coupled to the first module through a jump to the first original function.

32. The computerized system as in claim 31,

wherein the first data structure comprises a function lookup table readily available for verifying that the threaded local storage variable contains the correct original instrumented function address; and

wherein the second module comprises a dynamic linked library.

33. The computerized system as in claim 31, further comprising a second data structure associating the location of the first data structure with the location of the first module.

34. The computerized system as in claim 31, further comprising:

a third module of machine-readable code comprising:

a second instrumented function call to a second replacement function; and

a second data structure associating the identity of the second instrumented function with the location of the second instrumented function; and

a fourth module comprising the second replacement function, having a jump to the second original function.

35. The computerized system as in claim 31, further comprising:

a third module of machine-readable code comprising:

a second instrumented function jump to a second replacement function; and

a second data structure associating the identity of the second instrumented function with the location of the second instrumented function; and

wherein the second module further comprises the second replacement function, having a jump to the second original function.

36. A computer-readable medium having computer-executable instructions to a cause a computer to perform a method comprising:

redirecting an original function in an executable file to a user-supplied function; and

retaining access information of the original function.

37. A computer-readable medium having stored thereon a first data structure comprising a first module having a second data structure associating the identity of a first instrumented function with the location of the first instrumented function.

38. A computer-readable medium having stored thereon a first module comprising a first data structure associating the identity of a first instrumented function with the location of the first instrumented function.

39. The computer-readable medium having stored thereon a first module as in claim 38, wherein the first data structure further comprises a table.

40. The computer-readable medium having stored thereon a data structure as in claim 38, wherein the first data structure further comprises a threaded local storage variable.